



[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Login](#)

Search: ☒ The ACM Digital Library ☐ The Guide

SEARCH



[Feedback](#) [Report a problem](#) [Satisfaction survey](#)

Published before August 2001

Found 2 of 2

Sort results
by

relevance




Save results to a Binder

[Try an Advanced Search](#)

Try this search in The ACM Guide

Display results

expanded form 



Search Tips

☐ Open results in a new window

Results 1 - 2 of 2

Relevance scale ☐ ☒ ☐ ☐ ☐

¹ Reusable component interconnection patterns for distributed software architectures

Hassan Gomaa, Daniel A. Menascé, Michael E. Shin

May 2001 **ACM SIGSOFT Software Engineering Notes , Proceedings of the 2001 symposium on Software reusability: putting software reuse in context,** Volume 26 Issue 3

Full text available: pdf(181.77 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

This paper investigates the design of reusable component interconnection in client/ server systems. In particular, the paper describes the design of component interconnection patterns, which define and encapsulate the way client and server components communicate with each other. This paper uses the Unified Modeling Language (UML) to describe the component interconnection patterns for synchronous, asynchronous, and brokered communication. When designing a new distributed application, the app ...

Keywords: UML, client/server systems, distributed applications, patterns, software architecture, software component, software reuse

2 Network locality at the scale of processes

Jeffrey C. Mogul

August 1991 **ACM SIGCOMM Computer Communication Review , Proceedings of the conference on Communications architecture & protocols, Volume 21 Issue 4**

Full text available: pdf(1.19 MB) Additional Information: full citation, references, citings, index terms

Results 1 - 2 of 2

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2004 ACM, Inc.

[Terms of Usage](#) [Privacy Policy](#) [Code of Ethics](#) [Contact Us](#)

Useful downloads:  Adobe Acrobat  QuickTime  Windows Media Player  Real Player


[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Login](#)

 Search: ☒ The ACM Digital Library ☐ The Guide

 SEARCH

THE ACM DIGITAL LIBRARY


[Feedback](#) [Report a problem](#) [Satisfaction survey](#)

Published before August 2001

Found 1,464 of 1,464

Sort results by


[Save results to a Binder](#)
[Try an Advanced Search](#)

Display results


[Search Tips](#)
[Try this search in The ACM Guide](#)
☐ Open results in a new window

Results 1 - 20 of 200

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

Best 200 shown

Relevance scale ☐ ☐ ☐ ☐ ☐

1 [Assume-Guarantee Verification of Source Code with Design-Level Assumptions](#) May 2004 **Proceedings of the 26th International Conference on Software Engineering**

Full text available: pdf(147.92 KB)

Additional Information: [full citation](#), [abstract](#)
[Publisher Site](#)

Model checking is an automated technique that can be used to determine whether a system satisfies certain required properties. To address the "state explosion" problem associated with this technique, we propose to integrate assume-guarantee verification at different phases of system development. During design, developers build abstract behavioral models of the system components and use them to establish key properties of the system. To increase the scalability of model checking at this level, we have pr ...

2 [Oil and Water? High Performance Garbage Collection in Java with MMTk](#) May 2004 **Proceedings of the 26th International Conference on Software Engineering**

Full text available: pdf(183.10 KB)

Additional Information: [full citation](#), [abstract](#)
[Publisher Site](#)

Increasingly popular languages such as Java and C# require efficient garbage collection. This paper presents the design, implementation, and evaluation of MMTk, a Memory Management Toolkit for and in Java. MMTk is an efficient, composable, extensible, and portable framework for building garbage collectors. MMTk uses design patterns and compiler cooperation to combine modularity and efficiency. The resulting system is more robust, easier to maintain, and has fewer defects than monolithic collectors. ...

3 [Advanced control flow in Java card programming](#) Peng Li, Steve Zdancewic June 2004 **ACM SIGPLAN Notices , Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools**, Volume 39 Issue 7

Full text available: pdf(205.46 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Java Card technology simplifies the development of smart card applications by providing a high-level programming language similar to Java. However, the master-slave programming model used in current Java Card platform creates control flow difficulties when writing complex card programs, making it inconvenient, tedious, and error-prone to implement Java Card applications. This paper examines these drawbacks of the master-slave model and proposes a concurrent thread model for developing future Jav ...

Keywords: CPS, Java card, continuation, control flow, smart card, trampolined style

4 ESys.Net: a new solution for embedded systems modeling and simulation

J. Lapalme, E. M. Aboulhamid, G. Nicolescu, L. Charest, F. R. Boyer, J. P. David, G. Bois
June 2004 **ACM SIGPLAN Notices , Proceedings of the 2004 ACM SIGPLAN/SIGBED
conference on Languages, compilers, and tools**, Volume 39 Issue 7

Full text available:  pdf(276.12 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

The next generation of tools for embedded systems design will represent a common arena for several cooperating groups. These tools will permit system design at a high abstraction level and enable automatic refinement through several abstraction levels to obtain the final prototype. To facilitate this evolution, we propose a new .Net Framework based system level modeling and simulation environment. This environment allows (1) cooperation -- by enabling web-based design and multi-language features ...

Keywords: .Net, C#, C++, CIL, ESys.Net, HDLs, Java, SystemC, SystemVerilog, VHDL, Verilog, attribute programming, component-based programming, embedded systems, framework, hardware/software codesign, modeling, simulation, system on chip

5 DUX in practice I: Designing the handheld maritime communicator

Jesper Kjeldskov, Jan Stage
June 2003 **Proceedings of the 2003 conference on Designing for user experiences**

Full text available:  pdf(289.11 KB) Additional Information: [full citation](#), [abstract](#), [references](#)

We present the process of designing the first prototype of the Handheld Maritime Communicator: a mobile computer system supporting communication and coordination of safety-critical work activities on large container vessels. Designing the user experience of the Handheld Maritime Communicator was a particular challenge because it targets a highly specialized context of use and because poor design could potentially become a safety hazard. Meeting this challenge, ethnographic field studies on board ...

Keywords: collaborative work, field studies, handheld computing, iterative design, object-oriented analysis, safety-critical use domains, text-based communication, usability evaluation

6 Balancing register allocation across threads for a multithreaded network processor

Xiaotong Zhuang, Santosh Pande
June 2004 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2004 conference
on Programming language design and implementation**, Volume 39 Issue 6

Full text available:  pdf(429.85 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)


Modern network processors employ multi-threading to allow concurrency amongst multiple packet processing tasks. We studied the properties of applications running on the network processors and observed that their imbalanced register requirements across different threads at different program points could lead to poor performance. Many times application needs demand some threads to be more performance critical than others and thus by controlling the register allocation across threads one could impa ...

Keywords: multithreaded processor, network processor, register allocation

7 Cloning-based context-sensitive pointer alias analysis using binary decision diagrams

John Whaley, Monica S. Lam

June 2004 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation**, Volume 39 Issue 6

Full text available:  pdf(277.87 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)


This paper presents the first scalable context-sensitive, inclusion-based pointer alias analysis for Java programs. Our approach to context sensitivity is to create a clone of a method for every context of interest, and run a *context-insensitive* algorithm over the expanded call graph to get *context-sensitive* results. For precision, we generate a clone for every acyclic path through a program's call graph, treating methods in a strongly connected component as a single node. Normally ...

Keywords: Datalog, Java, binary decision diagrams, cloning, context-sensitive, inclusion-based, logic programming, pointer analysis, program analysis, scalable

8 Scalable lock-free dynamic memory allocation

Maged M. Michael

June 2004 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation**, Volume 39 Issue 6

Full text available:  pdf(213.94 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)


Dynamic memory allocators (malloc/free) rely on mutual exclusion locks for protecting the consistency of their shared data structures under multithreading. The use of locking has many disadvantages with respect to performance, availability, robustness, and programming flexibility. A lock-free memory allocator guarantees progress regardless of whether some threads are delayed or even killed and regardless of scheduling policies. This paper presents a completely lock-free memory allocator. It uses ...

Keywords: async-signal-safe, availability, lock-free, malloc

9 KISS: keep it simple and sequential

Shaz Qadeer, Dinghao Wu

June 2004 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation**, Volume 39 Issue 6

Full text available:  pdf(204.52 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)


The design of concurrent programs is error-prone due to the interaction between concurrently executing threads. Traditional automated techniques for finding errors in concurrent programs, such as model checking, explore all possible thread interleavings. Since the number of thread interleavings increases exponentially with the number of threads, such analyses have high computational complexity. In this paper, we present a novel analysis technique for concurrent programs that avoids this exponent ...

Keywords: assertion checking, concurrent software, model checking, program analysis, race detection

10 Race checking by context inference

Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar

June 2004 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation**, Volume 39 Issue 6

Full text available:  pdf(328.94 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Software model checking has been successful for *sequential* programs, where predicate abstraction offers suitable models, and counterexample-guided abstraction refinement permits the automatic inference of models. When checking *concurrent* programs, we need

to abstract threads as well as the contexts in which they execute. Stateless context models, such as predicates on global variables, prove insufficient for showing the absence of race conditions in many examples. We therefore use ...

Keywords: race conditions, software model checking

11 Evaluation: An improved slicer for Java

Christian Hammer, Gregor Snelting

June 2004 **Proceedings of the ACM-SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering**

Full text available:  pdf(180.49 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

We present an improved slicing algorithm for Java. The best algorithm known so far, first presented in [11], is not always precise if nested objects are used as actual parameters. The new algorithm presented in this paper always generates correct and precise slices, but is more expensive in general. We describe the algorithms and their treatment of objects as parameters. In particular, we present a new, safe criterion for termination of unfolding nested parameter objects. We then compare the two ...

Keywords: Java, object trees, static program slicing

12 Multiprocessor SoC MPSoC solutions/nightmare: Heterogeneous MP-SoC: the solution to energy-efficient signal processing

Tim Kogel, Heinrich Meyr

June 2004 **Proceedings of the 41st annual conference on Design automation**

Full text available:  pdf(74.19 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

To meet conflicting flexibility, performance and cost constraints of demanding signal processing applications, future designs in this domain will contain an increasing number of application specific programmable units combined with complex communication and memory infrastructures. Novel architecture trends like Application Specific Instruction-set Processors (ASIPs) as well as customized buses and Network-on-Chip based communication promise enormous potential for optimization. However, state-of- ...

Keywords: MP-SoC, design space exploration, energy efficiency, network-on-chip, signal processing

13 Multiprocessor SoC MPSoC solutions/nightmare: The future of multiprocessor systems-on-chips

Wayne Wolf

June 2004 **Proceedings of the 41st annual conference on Design automation**

Full text available:  pdf(58.72 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

This paper surveys the state-of-the-art and pending challenges in MPSoC design. Standards in communications, multimedia, networking, and other areas encourage the development of high-performance platforms that can support a range of implementations of the standard. A multiprocessor system-on-chip includes embedded processors, digital logic, and mixed-signal circuits combined into a heterogeneous multiprocessor. This mix of technologies creates a major challenge for MPSoC design teams. We will lo ...

Keywords: MPSoC, embedded software, low power, real-time, system-on-chip

14 ISSCC highlights: A dual-core 64b ultraSPARC microprocessor for dense server applications

Toshinari Takayanagi, Jinuk Luke Shin, Bruce Petrick, Jeffrey Su, Ana Sonia Leon
June 2004 **Proceedings of the 41st annual conference on Design automation**

Full text available:  pdf(645.83 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

A processor core, previously implemented in a 0.25 μ m Al process, is redesigned for a 0.13 μ m Cu process to create a dual-core processor with 1MB integrated L2 cache, offering an efficient performance/power ratio for compute-dense server applications. Deep submicron circuit design challenges, including negative bias temperature instability (NBTI), leakage and coupling noise, and L2 cache implementation are discussed.

Keywords: L2, NBTI, UltraSPARC, cache, coupling noise, deep submicron technology, dense server, dual-core, leakage, multiprocessor, negative bias temperature instability, throughput computing

15 ISSCC highlights: Design and implementation of the POWER5™ microprocessor

Joachim Clabes, Joshua Friedrich, Mark Sweet, Jack DiLullo, Sam Chu, Donald Plass, James Dawson, Paul Muench, Larry Powell, Michael Floyd, Balaram Sinharoy, Mike Lee, Michael Goulet, James Wagoner, Nicole Schwartz, Steve Runyon, Gary Gorman, Phillip Restle, Ronald Kalla, Joseph McGill, Steve Dodson

June 2004 **Proceedings of the 41st annual conference on Design automation**

Full text available:  pdf(422.45 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

POWER5 offers significantly increased performance over previous POWER designs by incorporating simultaneous multithreading, an enhanced memory subsystem, and extensive RAS and power management support. The 276M transistor processor is implemented in 130nm silicon-on-insulator technology with 8-level of Cu metallization and operates at >1.5 GHz.

Keywords: POWER5, clock gating, microprocessor design, power reduction, simultaneous multi-threading (SMT), temperature sensor

16 Design space exploration and scheduling for embedded software: High level cache simulation for heterogeneous multiprocessors

Joshua J. Pieper, Alain Mellan, JoAnn M. Paul, Donald E. Thomas, Faraydon Karim

June 2004 **Proceedings of the 41st annual conference on Design automation**

Full text available:  pdf(208.64 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

As multiprocessor systems-on-chip become a reality, performance modeling becomes a challenge. To quickly evaluate many architectures, some type of high-level simulation is required, including high-level cache simulation. We propose to perform this cache simulation by defining a metric to represent memory behavior independently of cache structure and back-annotate this into the original application. While the annotation phase is complex, requiring time comparable to normal address trace based sim ...

17 Memory and network optimization in embedded designs: An integrated hardware/software approach for run-time scratchpad management

Poletti Francesco, Paul Marchal, David Atienza, Luca Benini, Francky Catthoor, Jose M. Mendias

June 2004 **Proceedings of the 41st annual conference on Design automation**

Full text available:  pdf(137.98 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

An ever increasing number of dynamic interactive applications are implemented on portable consumer electronics. Designers depend largely on operating systems to map these

applications on the architecture. However, today's embedded operating systems abstract away the precise architectural details of the platform. As a consequence, they cannot exploit the energy efficiency of scratchpad memories. We present in this paper a novel integrated hardware/software solution to support scratchpad memories ...

Keywords: AMBA AHB, DMA, dynamic allocation, scratchpad

18 Tools and strategies for dynamic verification: Probabilistic regression suites for functional verification

Shai Fine, Shmuel Ur, Avi Ziv

June 2004 **Proceedings of the 41st annual conference on Design automation**

Full text available:  pdf(76.78 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Random test generators are often used to create regression suites on-the-fly. Regression suites are commonly generated by choosing several specifications and generating a number of tests from each one, without reasoning which specification should be used and how many tests should be generated from each specification. This paper describes a technique for building high quality random regression suites. The proposed technique uses information about the probability of each test specification coverin ...

Keywords: coverage analysis, functional verification, regression suite

19 Tools and strategies for dynamic verification: Industrial experience with test generation languages for processor verification

Michael Behm, John Ludden, Yossi Lichtenstein, Michal Rimon, Michael Vinov

June 2004 **Proceedings of the 41st annual conference on Design automation**

Full text available:  pdf(91.99 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

We report on our experience with a new test generation language for processor verification. The verification of two superscalar multiprocessors is described and we show the ease of expressing complex verification tasks. The cost and benefit are demonstrated: training takes up to six months; the simulation time required for a desired level of coverage has decreased by a factor of twenty; the number of escape bugs has been reduced.

Keywords: functional verification, processor verification, test generation

20 Crawling the web: Panorama: extending digital libraries with topical crawlers

Gautam Pant, Kostas Tsioutsoulis, Judy Johnson, C. Lee Giles

June 2004 **Proceedings of the 2004 joint ACM/IEEE conference on Digital libraries**

Full text available:  pdf(1.16 MB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

A large amount of research, technical and professional documents are available today in digital formats Digital libraries are created to facilitate search and retrieval of information supplied by the documents. These libraries may span an entire area of interest (e.g., computer science) or be limited to documents within a small organization. While tools that index, classify, rank and retrieve documents from such libraries are important, it would be worthwhile to complement these tools with infor ...

Keywords: classification, clustering, digital libraries, topical crawling, web mining

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2004 ACM, Inc.

[Terms of Usage](#) [Privacy Policy](#) [Code of Ethics](#) [Contact Us](#)

Useful downloads:  [Adobe Acrobat](#)  [QuickTime](#)  [Windows Media Player](#)  [Real Player](#)


[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Login](#)

 Search: ☒ The ACM Digital Library ☐ The Guide

 SEARCH

[Feedback](#) [Report a problem](#) [Satisfaction survey](#)

Published before August 2001

Found 1 of 1

Sort results by


[Save results to a Binder](#)
[Try an Advanced Search](#)
[Try this search in The ACM Guide](#)

Display results


[Search Tips](#)
☐ Open results in a new window

Results 1 - 1 of 1

Relevance scale ☐ ☐ ☐ ☐ ☐ ☒

1 [The design and performance of a real-time CORBA event service](#)

Timothy H. Harrison, David L. Levine, Douglas C. Schmidt

 October 1997 **ACM SIGPLAN Notices , Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications,**

Volume 32 Issue 10

Full text available: pdf(2.86 MB)

 Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

The CORBA Event Service provides a flexible model for asynchronous communication among objects. However, the standard CORBA Event Service specification lacks important features required by real-time applications. For instance, operational flight programs for fighter aircraft have complex real-time processing requirements. This paper describes the design and performance of an object-oriented, real-time implementation of the CORBA Event Service that is designed to meet these requirements. This paper ...

Results 1 - 1 of 1

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2004 ACM, Inc.

[Terms of Usage](#) [Privacy Policy](#) [Code of Ethics](#) [Contact Us](#)

 Useful downloads: [Adobe Acrobat](#) [QuickTime](#) [Windows Media Player](#) [Real Player](#)

IEEE HOME | SEARCH IEEE | SHOP | WEB ACCOUNT | CONTACT IEEE



Membership Publications/Services Standards Conferences Careers/Jobs

IEEE Xplore®
 RELEASE 1.8

 Welcome
 United States Patent and Trademark Office


» Se

[Help](#) [FAQ](#) [Terms](#) [IEEE Peer Review](#)
[Quick Links](#)

Welcome to IEEE Xplore®

- ☐ Home
- ☐ What Can I Access?
- ☐ Log-out

Tables of Contents

- ☐ Journals & Magazines
- ☐ Conference Proceedings
- ☐ Standards

Search

- ☐ By Author
- ☐ Basic
- ☐ Advanced
- ☐ CrossRef

Member Services

- ☐ Join IEEE
- ☐ Establish IEEE Web Account
- ☐ Access the IEEE Member Digital Library

IEEE Enterprise

- ☐ Access the IEEE Enterprise File Cabinet

Print Format

 Your search matched **724** of **1088345** documents.

 A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance Descending** order.

Refine This Search:

You may refine your search by editing the current search expression or entering a new one in the text box.

☐ Check to search within this result set

Results Key:

JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

1 SMTA: next-generation high-performance multi-threaded processor
Tu, J.-F.; Wang, L.-H.;

Computers and Digital Techniques, IEE Proceedings- , Volume: 149 , Issue: 5 , Sept. 2002

Pages:213 - 218

[\[Abstract\]](#) [\[PDF Full-Text \(624 KB\)\]](#) **IEE JNL**
2 On the performance of a multi-threaded RISC architecture
Lindsay, S.K.; Preiss, B.R.;

Electrical and Computer Engineering, 1993. Canadian Conference on , 14-17 Sept. 1993

Pages:369 - 372 vol.1

[\[Abstract\]](#) [\[PDF Full-Text \(368 KB\)\]](#) **IEEE CNF**
3 Multi-threading and remote latency in software DSMs
Thitikamol, K.; Keleher, P.;

Distributed Computing Systems, 1997., Proceedings of the 17th International Conference on , 27-30 May 1997

Pages:296 - 304

[\[Abstract\]](#) [\[PDF Full-Text \(1084 KB\)\]](#) **IEEE CNF**
4 Applying static analysis to large-scale, multi-threaded Java program
Artho, C.; Biere, A.;

Software Engineering Conference, 2001. Proceedings. 2001 Australian , 27-28 2001

Pages:68 - 75

[\[Abstract\]](#) [\[PDF Full-Text \(680 KB\)\]](#) IEEE CNF

5 Detailed design and evaluation of redundant multi-threading alternatives

Mukherjee, S.S.; Kontz, M.; Reinhardt, S.K.;

Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on , 25-29 May 2002

Pages:99 - 110

[\[Abstract\]](#) [\[PDF Full-Text \(419 KB\)\]](#) IEEE CNF

6 A multi-threading model for distributed mobile objects and its realization in FarGo

Abu, M.; Ben-Shaul, I.;

Distributed Computing Systems, 2001. 21st International Conference on. , 16 April 2001

Pages:313 - 321

[\[Abstract\]](#) [\[PDF Full-Text \(768 KB\)\]](#) IEEE CNF

7 Optimistic recovery in multi-threaded distributed systems

Damani, O.P.; Tarafdar, A.; Garg, V.K.;

Reliable Distributed Systems, 1999. Proceedings of the 18th IEEE Symposium on , 19-22 Oct. 1999

Pages:234 - 243

[\[Abstract\]](#) [\[PDF Full-Text \(128 KB\)\]](#) IEEE CNF

8 Design and implementation of multi-threaded object request broker

Yue-Shan Chang; Lo, W.; Chii-Jet Wang; Shyan-Ming Yuan; Liang, D.;

Parallel and Distributed Systems, 1998. Proceedings., 1998 International Conference on , 14-16 Dec. 1998

Pages:740 - 747

[\[Abstract\]](#) [\[PDF Full-Text \(120 KB\)\]](#) IEEE CNF

9 Probabilistic noninterference for multi-threaded programs

Sabelfeld, A.; Sands, D.;

Computer Security Foundations Workshop, 2000. CSFW-13. Proceedings. 13th IEEE , 3-5 July 2000

Pages:200 - 214

[\[Abstract\]](#) [\[PDF Full-Text \(440 KB\)\]](#) IEEE CNF

10 An MFC based multi-threaded test environment for the validation of embedded automotive microcontroller

Khwaja, A.A.;

Technology of Object-Oriented Languages and Systems, 2000. TOOLS 34. Proceedings. 34th International Conference on , 30 July-4 Aug. 2000

Pages:15 - 24

[\[Abstract\]](#) [\[PDF Full-Text \(540 KB\)\]](#) IEEE CNF

11 MTIO. A multi-threaded parallel I/O system*More, S.; Choudhary, A.; Foster, I.; Xu, M.Q.;*

Parallel Processing Symposium, 1997. Proceedings., 11th International , 1-5 / 1997

Pages:368 - 373

[\[Abstract\]](#) [\[PDF Full-Text \(544 KB\)\]](#) IEEE CNF**12 Variability in architectural simulations of multi-threaded workloads***Alarneldeen, A.R.; Wood, D.A.;*The Ninth International Symposium on High-Performance Computer Architectu
2003. HPCA-9 2003. Proceedings. , 8-12 Feb. 2003

Pages:7 - 18

[\[Abstract\]](#) [\[PDF Full-Text \(380 KB\)\]](#) IEEE CNF**13 Real-time garbage collection in multi-threaded systems on a single processor***Siebert, F.;*

Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE , 1-3 Dec.

Pages:277 - 278

[\[Abstract\]](#) [\[PDF Full-Text \(20 KB\)\]](#) IEEE CNF**14 Synchronization and control of multi-threads for MPEG-4 video dec***Xuemin Chen;*

Consumer Electronics, 1999. ICCE. International Conference on , 22-24 June

Pages:298 - 299

[\[Abstract\]](#) [\[PDF Full-Text \(120 KB\)\]](#) IEEE CNF**15 SMT layout overhead and scalability***Burns, J.; Gaudiot, J.-L.;*Parallel and Distributed Systems, IEEE Transactions on , Volume: 13 , Issue:
2 , Feb. 2002

Pages:142 - 155

[\[Abstract\]](#) [\[PDF Full-Text \(514 KB\)\]](#) IEEE JNL

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#) [17](#) [18](#) [19](#) [20](#) [21](#) [22](#) [23](#)
[25](#) [26](#) [27](#) [28](#) [29](#) [30](#) [31](#) [32](#) [33](#) [34](#) [Next](#)

[Home](#) | [Log-out](#) | [Journals](#) | [Conference Proceedings](#) | [Standards](#) | [Search by Author](#) | [Basic Search](#) | [Advanced Search](#) | [Join IEEE](#) | [Web Account](#) |
[New this week](#) | [OPAC Linking Information](#) | [Your Feedback](#) | [Technical Support](#) | [Email Alerting](#) | [No Robots Please](#) | [Release Notes](#) | [IEEE Online](#)
[Publications](#) | [Help](#) | [FAQ](#) | [Terms](#) | [Back to Top](#)

Copyright © 2004 IEEE — All rights reserved

[IEEE HOME](#) | [SEARCH IEEE](#) | [SHOP](#) | [WEB ACCOUNT](#) | [CONTACT IEEE](#)

[Membership](#) | [Publications/Services](#) | [Standards](#) | [Conferences](#) | [Careers/Jobs](#)
IEEE Xplore
RELEASE 1.8

 Welcome
 United States Patent and Trademark Office

[Help](#) | [FAQ](#) | [Terms](#) | [IEEE Peer Review](#)
[Quick Links](#)
Welcome to IEEE Xplore®

- ☐ Home
- ☐ What Can I Access?
- ☐ Log-out

Tables of Contents

- ☐ Journals & Magazines
- ☐ Conference Proceedings
- ☐ Standards

Search

- ☐ By Author
- ☐ Basic
- ☐ Advanced
- ☐ CrossRef

Member Services

- ☐ Join IEEE
- ☐ Establish IEEE Web Account
- ☐ Access the IEEE Member Digital Library

IEEE Enterprise

- ☐ Access the IEEE Enterprise File Cabinet

 Your search matched **0** of **1088345** documents.

 A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance Descending** order.

Refine This Search:

You may refine your search by editing the current search expression or entering a new one in the text box.

☐ Check to search within this result set

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

Results:
No documents matched your query.
Print Format
[Home](#) | [Log-out](#) | [Journals](#) | [Conference Proceedings](#) | [Standards](#) | [Search by Author](#) | [Basic Search](#) | [Advanced Search](#) | [Join IEEE](#) | [Web Account](#) | [New this week](#) | [OPAC Linking Information](#) | [Your Feedback](#) | [Technical Support](#) | [Email Alerting](#) | [No Robots Please](#) | [Release Notes](#) | [IEEE Online Publications](#) | [Help](#) | [FAQ](#) | [Terms](#) | [Back to Top](#)

Copyright © 2004 IEEE — All rights reserved

IEEE HOME | SEARCH IEEE | SHOP | WEB ACCOUNT | CONTACT IEEE



Membership Publications/Services Standards Conferences Careers/Jobs

IEEE Xplore®
 RELEASE 1.8

 Welcome
 United States Patent and Trademark Office


» Se.

[Help](#) [FAQ](#) [Terms](#) [IEEE Peer Review](#)
[Quick Links](#)

Welcome to IEEE Xplore®

- ☐ Home
- ☐ What Can I Access?
- ☐ Log-out

Tables of Contents

- ☐ Journals & Magazines
- ☐ Conference Proceedings
- ☐ Standards

Search

- ☐ By Author
- ☐ Basic
- ☐ Advanced
- ☐ CrossRef

Member Services

- ☐ Join IEEE
- ☐ Establish IEEE Web Account
- ☐ Access the IEEE Member Digital Library

IEEE Enterprise

- ☐ Access the IEEE Enterprise File Cabinet

Your search matched **1** of **1088345** documents.A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance Descending** order.

Refine This Search:

You may refine your search by editing the current search expression or entering a new one in the text box.

☐ Check to search within this result set

Results Key:

JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

**1 Transaction management for a distributed object storage system
WAKASHI-design, implementation and performance**

Ge Yu; Kaneko, K.; Bai, G.; Makinouchi, A.;

Data Engineering, 1996. Proceedings of the Twelfth International Conference on , 26 Feb.-1 March 1996

Pages:460 - 468

[\[Abstract\]](#)
[\[PDF Full-Text \(1000 KB\)\]](#)
IEEE CNF

Print Format

[Home](#) | [Log-out](#) | [Journals](#) | [Conference Proceedings](#) | [Standards](#) | [Search by Author](#) | [Basic Search](#) | [Advanced Search](#) | [Join IEEE](#) | [Web Account](#) | [New this week](#) | [OPAC Linking Information](#) | [Your Feedback](#) | [Technical Support](#) | [Email Alerting](#) | [No Robots Please](#) | [Release Notes](#) | [IEEE Online Publications](#) | [Help](#) | [FAQ](#) | [Terms](#) | [Back to Top](#)

Copyright © 2004 IEEE — All rights reserved

Transaction Management for a Distributed Object Storage System WAKASHI

– Design, Implementation and Performance

Ge Yu*, Kunihiko Kaneko, Guangyi Bai, Akifumi Makinouchi
Department of Computer Science and
Communication Engineering,
Kyushu University, Japan

Abstract

This paper presents the transaction management in a high performance distributed object storage system WAKASHI. Unlike other systems that use centralized client/server architecture and offer conventional buffer management for distributed persistent object management, WAKASHI is based on symmetric peer-peer architecture and employs memory-mapping and distributed shared virtual memory techniques. Several novel techniques on transaction management for WAKASHI are developed. First, a multi-threaded transaction manager offers "multi-threaded connection" so that data control and transaction operations can be performed in parallel manner. Secondly, a concurrency control mechanism supports transparent page-level locks to reduce the complexity of user programs and locking overhead. Thirdly, a "compact commit" method is proposed to minimize the communication cost by reducing the amount of data and the number of connections. Fourthly, a redo-only recovery method is implemented by "shadowed cache" method to minimize the logging cost, and to allow fast recovery and system restart. Moreover, the system offers "hierarchical" control to support nested transactions. A performance evaluation by the OO7 benchmark is presented, as well.

1 Introduction

Much effort has been made to develop database systems that support advanced applications such as CAD/CAM, CASE[1][8][14]. Typical advanced applications are complex tasks that deal with complex objects. In these applications, user transactions are also complex to represent co-operative activities, and may access several databases located on different sites in a distributed environment connected by network. The problems on complex object management are well solved by using the object-orientation paradigm in many commercial systems like O2 [9] and ObjectStore[15]. However, not all problems have been solved on transaction management. Most systems adopt a centralized approach by which a server acts as a centralized DBMS to perform centralized control on data and transactions. The centralized systems have

difficulties to support more complex applications in that : (1) as the number of sites increases, the server will become bottleneck for heavy workload; (2) a centralized system can not manage distributed databases effectively. On the other hand, with decreasing cost of powerful workstations, the differences between client machines and server machines disappear because every machine can have equal functions. A new trend of the system architecture is peer-peer architecture. Some prototype systems like SHORE and Bess adopted this type of architecture[6][4]. However, these systems focused on object management. The issues about their transaction management were briefly described as by employing the traditional techniques. The impacts of new software and hardware techniques on transaction management were not deeply studied.

We think that transaction management in a distributed object storage system can benefit from the new resources, such as multi-processors and massive memory, which allow to build a high performance transaction management system. In this paper, we will discuss how to achieve this target on the five issues as follows:

(1) The structure of transaction manager. Although thread facilities can be exploited for increasing execution concurrency, most storage systems deal with the requests from a transaction by one thread no matter these requests may be processed in parallel[8][14]. We call it as "single-threaded connection" approach. We built a transaction manager that can offer several threads for a transaction so that database control and transaction operations such as abort and commit can be performed in parallel. We call this new approach as "multi-threaded connection" approach.

(2) The concurrency control and cache coherency control mechanism. Most systems require explicit interactions between clients and servers, for example, sending a command to request a page, or to lock an object[1][9]. Their control protocol and mechanism are implemented on this explicit interface. This approach incurs the complexity of user programs and locking overhead. By virtual memory mapping techniques, we offer transparent locks for user programs. The access on conflicted data or invalid data is automatically detected, and then the required control is

*On leave from Northeastern University, China

(3) The remote transaction commit. Most systems transfer updated pages from a client to a server during executing transactions [6][15]. This approach incurs the heavy communication cost and unnecessary operations in some case. For example, if the transaction is aborted later, the transferring cost is wasted. We designed a "compact commit" method by which the results of a transaction are transferred in one or a few package after it is committed. This can minimize the communication cost by reducing the amount of data and the number of connections.

(5) The transaction model. Most systems only support ordinary transactions that are "flat"[4][6]. We support nested transactions for advanced applications[17]. Nested transactions have powerful modeling ability to represent a complex work by some decomposable subtransactions and offer finer-grained control over concurrency and recovery than "flat" transactions. In addition, nested transactions facilitate potential parallelism in applications, that can be exploited to improve system performance in a distributed multi-processor workstation environment. The management on nested transactions is in "hierarchical" style unlike "flat" style of ordinary transactions. Therefore, our transaction management is designed to offer a "hierarchical" control on transaction execution. This distinguishes our control mechanism from "flat" mechanisms of other systems.

The paper will deal with these issues by the design and implementation of transaction management for WAKASHI, which is a distributed object storage system developed at Kyushu University[2]. The main goal of WAKASHI is to provide rapid and transparent access on global volatile/persistent objects without explicit control on the object locations and object status, as well as to support consistency and reliability of distributed databases.

The rest of this paper is organized as follows. Section 2 provides an overview of the WAKASHI peer-peer architecture and describes the basic concepts of nested transactions, the concurrency and cache coherency control protocol, and the logging method. Transaction operations including normal processing, commit and abort, are presented in Section 3. The recovery management for site crash and network crash is discussed in Section 4. The performance experiments

2 Architecture Overview

[illegible]

In WAKASHI, each server is a repository of the database and the log file. The basic storage unit is a *heap*. The *heap* offers storage for objects without limitation on size and structure¹. Each heap is a collection of pages. Each page has a sequential number in a heap as its identifier (denoted as *pid*). The heap that can be shared in a network is called *global heap*. The global heap can be persistent or volatile. A heap becomes persistent after saved in a file and a persistent object is stored in a heap file. WAKASHI databases are collections of such persistent heap files. In this paper, we only consider global persistent heaps and will refer heaps as global persistent heaps sometimes.

After a heap file is opened by a client at a site, the mapping between the client and the server, the mapping between the server and the disk, and the mapping between the server and the server are created. Hence, a heap may have many replicas in a network. For a heap, the original site where it is stored is called its *primary site*, and the other sites that have a replica of it are called its *mirror sites*. For example, the heap H

461

in Fig.1 has site A as its *primary site* and site B as its *mirror site*.

For each heap, a server maintains a server cache in its virtual memory that has physical memory used as physical cache, and each client also maintains a client cache in its own virtual memory that is a mapping from the server. To do this, a heap file is mapped into the server virtual memory and the client virtual memory, respectively, using UNIX `mmap()` function with `map_shared` mode. This makes all clients at a site share the same physical cache of the server and don't occupy extra storage space. At a mirror site like site B in Fig.1, a temporary heap file is used as the swap space of the memory mapping. This file also has functions of local disk caching[10].

At each site, there are at least one server and several clients. An application is generated as a client by linking its application program with the WAKASHI client library. An application may consist of many transactions, moreover, they may be nested. Since objects in a heap are mapped into a client virtual memory, transactions in the client can directly manipulate these objects in a transparent way. All desirable controls are automatically performed by the server.

2.1 Process Structure

In WAKASHI, both clients and servers are implemented as multi-threaded processes. In other systems [8][14], although a client may access several database files, its requests are represented at most one thread in the server thus at any time the connection between a client and the server is "single-threaded". In addition, they schedule threads of different transactions in serial mode because the control information such as lock table is managed in a centralized mode. To exploit parallelism in a multi-processor environment, we create thread for each accessed heap in a client, and create a thread for each opened heap in a server. Thus there is a "multi-threaded" connection between the client and the server when the client accesses more than one heap. Moreover, the control information are distributed on different heaps. Thus, the control on different heaps can be performed in parallel.

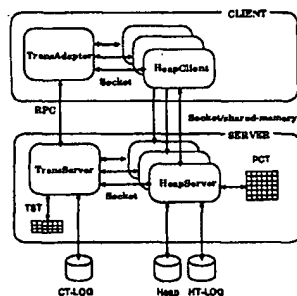


Figure 2: Process Structure at a Site

Fig.2 illustrates the internal process structure between a client and its server. In a client, each HeapClient is a thread for access on a heap, and

TransAdapter is the coordinator process for the threads of the client and the interface for transaction operations. In a server, each HeapServer is a thread that performs control and provides services on a heap, and TransServer is the coordinator process for the threads of the server and the transaction manager. The main data structures are also shown in Fig.2. PCT is a *page control table* that keeps the information for concurrency control and cache coherency control. TST is a *transaction state table* that keeps the information about currently executing transactions. Two types of log files are maintained, CT-Log is for logging transaction states(begin, abort or commit), and HT-Log is for logging update operations of transactions. These data structures will be detailed later. Communication is implemented in several ways(e.g. socket, shared-memory, and RPC) on the concerns of performance and communication cost.

WAKASHI offers clients a transparent way for remote access by DSM approach. All necessary actions on DSM control(mainly cache coherency control) are automatically performed by servers. The control on a *primary heap* and a *mirror heap* is performed by the two threads on behalf of them so that these operations do not impede the operations on other heaps.

2.2 Nested Transaction

In WAKASHI, we adopted the nested transaction model introduced by Moss[17]. In this model, a nested transaction is formed in a tree structure. The root is called *root transaction*. A transaction that encloses subtransactions are called *parent*, and the enclosed subtransactions are called its *children*. The children with the same parent are called *sibling*. In a path from the root to the leaf, the transactions in the superior positions are *ancestors* and the transactions in the inferior positions are *descendants*. Those relationships determine the control on the execution of each subtransaction. In this paper, subtransactions in a nested transaction are also called transactions.

Although nested transactions were discussed in many papers, they were implemented in few systems [15]. In WAKASHI, we implemented nested transactions by multi-thread and virtual memory techniques. A client starts transaction by a *begin* command and terminates a transaction by a *commit* or an *abort* command. A child transaction is started if its *begin* command is nested in another transaction. A child transaction must be terminated before its parent. Each transaction has an identifier(denoted as *tid*). The transaction information is kept in TST table at each site as shown in Fig. 2. Each entry of TST contains the *tid*, type, parent, and state of a running transaction.

To support parallel transaction execution, we designed a two-layer scheme by exploiting multi-thread facilities. The two-layers are the internal view from the transaction manager. The upper layer is called *client transaction (CT)* and the lower layer is called *heap transaction(HT)*. For example, if a transaction T operates two heaps, then T consists of a CT and two HTs. CT contains transaction dependency semantics and HT contains database operation seman-

tics. The transaction management of WAKASHI is based on this two-layer execution model. In Fig.2, TransAdapter and TransServer perform operations for CTs, and HeapClient and HeapServer perform operations for HTs.

From the viewpoint of a server, each client has only one active nested transaction and executes only one subtransaction at a time. For a running nested transaction, at any time only one path starting from the root transaction is active. We call this path as *active path*. Actually, more higher degree parallelism, like sibling parallelism or parent/child parallelism, is possible for a nested transaction. It can be implemented with several parallel processes by an application server above WAKASHI. As a physical level, WAKASHI only deals with "single-process" execution and offers low-level parallelism. However, the control protocol and algorithm can also be employed to support "multiple-process" execution of nested transactions.

2.3 Concurrency Control

In WAKASHI, concurrent access to a heap file is controlled by using page-level locks. We first describe implementation of lock mechanism and cache coherency. As stated above, locking operations are automatic and transparent for transactions. They are implemented by the virtual memory protection mechanisms provided by hardware. We use three protection modes for a page P in a virtual memory: (1)none: P is invalid, (2)read: P is read-allowed, and (3)write: P is read-allowed and write-allowed.

After a heap in the server cache is mapped into a client, the memory protection on each page of the heap in the client cache is setup to "none" by `mprotect()` function. At the beginning of a transaction T of the client, the protection on each page is usually "none". When T tries to access a page P with an inaccessible protection mode such as "none", a signal (called *pagefault*) must occur. The interrupt handler of the client thread (HeapClient) traps the signal and sends the message (tid, pid and access type) to the corresponding server thread (HeapServer) that in turn verifies the request against the PCT. After granting the lock and ensuring that P is valid, the HeapServer tells the HeapClient to set the memory protection of P with "read" or "write" attribute correctly. In this way, T can access P without explicit locking commands. After a lock on P is released by T, the memory protection of P is changed back to "none".

For coherency control on DSM, we employ callback protocol because it has good performance over a wider range of workloads[20]. By callback protocol, the lock on a cached page is still retained by a site even if no transaction locks it at the site. The site doesn't need to contact other sites to check page validity or to acquire a lock when a transaction accesses a cached page with the appropriate lock. Therefore, a lock can be granted at the local site in some cases. Only when the write lock on a page is requested, all replicas of the page on different sites are invalidated (called *callback*).

Next, we introduce the lock protocol. In WAKASHI, we designed an extended two-phase locking protocol called N-2PL. It is an extension of strict

2PL to support locking inheritance of nested transactions, and to support cache coherency control required by DSM. By N-2PL, subtransactions of a nested transaction acquire locks on pages when they first manipulate them, and these locks will be held or retained until the root transaction is terminated (committed or aborted). Note that those locks that are not inherited from its ancestors will be released if the descendant is aborted.

N-2PL supports two-directional locking inheritance: *upward inheritance* and *downward inheritance*. The former means that all locks of a child are delegated to its parent after it is terminated; and the latter means that a descendant can get all locks that are held or retained by any of its ancestors. In WAKASHI, the performance of lock acquiring can gain from the virtual memory techniques. Because a descendant runs in the same virtual memory as its ancestors, its read operations will not trigger pagefault on a page that has been locked by any of its ancestors so that the overhead of lock-request and lock-release is minimized.

As stated in subsection 2.1, each heap has a PCT that keeps lock information, and lock acquiring is based on PCT. The initial locking status of a page P is free. P can be locked by transactions in one of four status: read, write, read-retain, and write-retain. In a PCT entry of P, the set *users* contains either a writer or all readers of P; the list *waiters* contains all requesters of P that are blocked; the stack *retainers* contains all retainers in an active path; and the stack *beforeImages* contains the before images of P updated in an active path.

The last issues to be discussed is about deadlock. As we know, lock-based protocols may incur deadlock due to cyclically waiting of blocked transactions. To solve the deadlock, a deadlock detection mechanism is setup. However, the cost of deadlock detection for nested transactions is very high due to the complex structure and relationship[12]. Not only wait-for-lock information but also wait-for-commit information are needed to find a cycle. To reduce the complexity and the overhead on the deadlock detection, we designed N-2PL protocol to be deadlock free in a client. In our protocol, the downward inheritance is in unconditional mode unlike [12] that could cause blocking. Let us consider a running nested transaction. The transaction that is not in the active path is either not invoked yet or already terminated. This means there is no cycle between the transactions inside the path and those outside the path. On the other hand, the transaction that is in the active path can always get lock from its parent or ancestors. This means that no blocking occurs between the transactions in the path. Therefore, the nested transaction in a client is deadlock free by our scheme. One problem of un-controlled downward inheritance is that the updated pages of ancestors can not be protected from its descendants. To overcome this problem, operations on these pages need to be defined as separate subtransactions by users.

In WAKASHI, the deadlock detection employs waits-for graph (WFG) approach[3]. When a lock request is rejected, a WFG is reconstructed. Since the conflicts are always related with different clients, the

detection algorithm is the same as the usual one[3].

2.4 Recovery

To simplify recovery processing and improve recovery efficiency, we implemented a "redo-only" recovery method. In this subsection, the basic ideas about our recovery method are introduced and the detail will be described in section 4.

First, the "shadowed cache" method is described. In an ordinary redo-only recovery method [3], the updated data of an uncommitted transaction are put in an intention list from which new data are read by the transaction. This method incurs extra overhead for data access operations. Using virtual memory techniques, we solved this problem by allowing update operations to be performed in the server cache in "update-in-place" manner so that there are no extra overheads on subsequent operations. However, since we can not control the swapping operations of UNIX cache management, the updated pages of uncommitted transactions may be swapped into database files. At a mirror site, there is no problem because the server cache is mapped to a local file. At the primary site, the problem would occur if clients shared the server cache of the database file (called *primary cache*). To avoid undesirable swapping, we create another server cache that can be shared by clients at the primary site. This cache uses a temporary file as swapping space. We call this cache as "shadowed cache". The data in a "shadowed cache" is fetched from the primary cache by "copy-on-read" method when the cache is empty. It seems that we could solve the swapping problem by "copy-on-write" mapping mode of `mmap()`. We found that this method is not suitable because it allocates an extra private space from the system swapping area. When the size of data files or the number of clients grows, the system swapping space would be overflowed. The cache at a mirror site can also be considered as a "shadowed cache".

Next, we describe undo operations for transactions. The before image of an updated page is saved by using a "backup page" method. When a write pagefault on page P occurs, the before image of P is saved into a backup page that is allocated in the virtual memory. Since the downward inheritance of lock acquiring, P may have several before images. To keep track the updates of a child transaction, all write-protection attributes in the client virtual memory must be changed into "read" before it begins. When a transaction is aborted, P can be restored immediately from its backup page. When a transaction is committed, each of its backup pages is freed if it is root or its parent already has the backup page, otherwise, the backup page is delegated to its parent.

Finally, the structures of log files and log records are briefly described. In terms of the two-layer execution model of transactions, two kinds of log file are maintained: CT-Log and HT-Log. This facilitates parallel logging operations and recovery. Most systems only provide one or a few log files[11][16], which would become a bottleneck for transaction execution. For logging a transaction, the CT-Log file at its execution site, and the HT-Log files at the primary sites

are used.

In a CT-Log file, a *begin* record contains the tid of a started transaction, and the path names of heap files that are accessed; a *commit* record contains the tid of a committed transaction; and an *abort* record contains the tid of an aborted transaction. In an HT-Log file, a *start* record contains the tid of a transaction that updates this heap; an *end* record contains the tid of a transaction that finishes operations; and a *redo* record contains the tid of an updating transaction, the pid and the compressed form of differential block of the updated page. In addition, CT-Log file has a *checkpoint* record. A checkpoint is generated periodically or when the server is idle. At checkpoint time, the dirty pages of committed transactions are flushed into disk in the commit sequence of the transactions. The dirty page is determined by the HT-Log and `mincore()` function. The flush is done by using `msync()` function. Then, a *checkpoint* record is written into disk. It contains the position of the oldest transaction that has not been forced.

3 Transaction Execution

In this section, we further discuss the concurrency control, cache coherency control and recovery through the execution of a nested transaction. The state transition diagram of a root transaction T is shown in Fig. 3, where an arrow means the state transition and a message attached on the arrow is the internal command causing such state transition. T is first in *null* state. After T is initialized by a *begin* command, T goes into *activated* state. In *activated* state, T may have three sub-states: when an operation of T is being performed, T goes into *executed* states; when a new subtransaction is started, T goes into *sync* state; when a locking request is rejected, T goes into *blocked* state. When T finishes all operations and all its subtransactions, T goes into *prepared* state. In *prepared* state, if T can be committed in terms of the commit protocol, T goes into *committed* state. If T is aborted by the system, or by itself in *activated* state or *prepared* state, T goes into *aborted* state. In addition, if T can not determine to commit or abort in *prepared* state, it goes into *frozen* state (this will be detailed later). After T finishes commit or abort, T goes into *terminated* state.

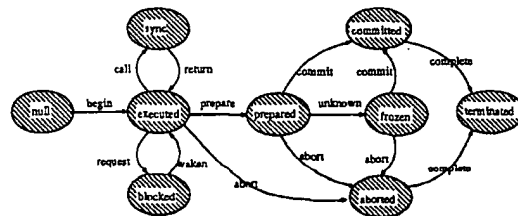


Figure 3: Execution State of Transaction

3.1 Normal Processing

From the beginning of a transaction to the finishing of the last operation is the normal processing. In an

ordinary system, two tasks are to be done by transaction management during normal processing: one is the schedule of blocked transactions due to concurrency control, the other is the log generation due to update operations. Because update operations are performed in "shadowed cache" manner, no more operations except for generating backup pages are done in the normal processing in WAKASHI.

For lock acquiring in a distributed system, there are many methods such as *primary site* and *primary copy*[3]. In WAKASHI, we extend the primary site approach by the callback protocol of DSM. Let us illustrate the locking processing by two examples which show how the callback function is integrated. We denote the primary site of a heap *H* as *PSite*, and the mirror site of *H* as *MSite*. Suppose *P* is a page of *H*. Example 1: Read Lock Request. Consider a client at a mirror site *MSite1* that tries to read *P*. First, *MSite* applies for the read lock from *PSite*. If *P* is currently written by another mirror site *MSite2*, then the request is blocked. After *MSite2* releases the write lock on *P*, a response is sent to *PSite* thus *PSite* gets the read lock from *MSite2*. *PSite* in turn sets the read lock on *P* for *MSite1* and sends a response to *MSite1*. In addition, if *P* has been updated, the new data of *P* will be transferred to *MSite1*.

Example 2: Write Lock Request. Consider a client at a mirror site *MSite1* that tries to write *P*. First, *MSite1* sends a write lock request to *PSite*. Before *MSite1* can get the write lock on *PSite*, *PSite* forwards the write lock request to all *MSites* that are readers of *P*. After *PSite* gets the write lock on *P* from all the other sites, it sets the write lock on *P* for *MSite1* and sends a response to *MSite1*.

Before a client *C* gets a lock, *C* must wait until a response comes. After the page lock is acquired, the server sends *C* a response message to re-schedule *C*. In addition, after a site releases a write lock on *P*, it retains a read lock on *P* because it still has a valid copy. This read lock is also used for callback by the primary site later.

3.2 Transaction Commit

In WAKASHI, the processing on transaction commit includes that on a subtransaction and on a root transaction. The commit of a subtransaction performs no operations except for delegating all its locks and backup pages to its parent. The commit of a root transaction involves the distributed operations if it updates at least one remote heap. The most popular protocol for distributed commit is *2-phase-commit*(2PC) protocol[5]. However, we can not apply 2PC directly in WAKASHI because the implementation of an ordinary 2PC only involves the short control messages while the commit activity in a distributed object storage system involves long messages used for page transferring. Therefore, We adopted two methods to optimize 2PC when applying it in WAKASHI.

The first method is "compact commit" technique. Unlike ordinary systems that generate the log records one by one for each update operation of transactions[11][16], we generate the log records after finishing all operations of a nested transaction and

send them in one or a few long packages. It has two advantages: (1) to write log files in block mode is faster than in record mode. For example, the I/O cost of writing 10 records in one block is almost the same as that of writing only one record in one block. (2) the number of log records is greatly reduced, which is equal to the number of updated pages in our method, otherwise, equal to the number of update operations by ordinary methods. Since the size of log information and the number of messages are reduced, the method can greatly reduce the communication cost when the log records are shipped from a mirror site to the primary site.

The second method is the "redo-at-primary-site" technique. Other systems[15][16] transfer the new pages from client to a remote server. Instead, we just transfer the log to the primary site and redo the log upon the *primary cache* after the transaction is committed.

In WAKASHI, 2PC is implemented in the following way. Suppose *T* is to be committed. The server at the execution site of *T* acts as the coordinator, and the servers at other primary sites acts as participants. In *preparing phase* of 2PC, the log records are generated at the execution site, and transferred to the primary site where they are then flushed into disk; In *decision phase* of 2PC, the log is redone upon the *primary cache* at the primary site if a commit command is received. After a root transaction is committed successively, all write locks of it are released and a *commit* record is written in CT-log.

3.3 Transaction Abort

The basic processing steps to abort a transaction are: (1) release all read locks; (2) if it is a root transaction, write an *abort* record on CT-Log; (3) for each updated page *P*, undo *P*, and retain or release the write lock.

To abort a transaction *T* at a mirror site or at its primary site is same. Only the shadowed cache in which *T* is executed is to be undone. The operations of *T* are undone by restoring the before image of each updated page of *T* from its backup page. WAKASHI provides an efficient way of undoing a nested transaction. If *T* is to be aborted, the sub-tree whose root is *T* is also to be aborted. We call this sub-tree as *aborting tree*. In ordinary method, each subtransaction in the aborting tree must be undone one by one by using a recursive algorithm[19]. By our "backup page" method, we need only undo once for each updated page with the oldest before image starting from *T* no matter how many times the page has been updated by the subtransactions of *T*.

4 Crash Recovery

For a distributed system, there are several types of crash due to software or hardware failure. The first type is site crash, that may occur when a site fails. The second type is network crash, that may occur when a link between two sites fails. In addition, a site may be crashed by a media failure such as damage of disks. The media failure is not discussed in this paper because it can only be solved by archival copy[3].

Since WAKASHI adopts redo-only approach and 2PC protocol, it ensures that each database file doesn't contain updated pages of uncommitted transactions, and all database files are updated consistently. The only task of the crash recovery is to redo the transactions that are already committed before crash.

4.1 Site Crash

When a site *S* is crashed, a transaction *T* at *S* may be in different states.

Case 1: If *T* is in *activated* state, the database files have not been updated.

Case 2: If *T* is in *prepared* state, some HT-Log files have been written and some have not, but the database files still have not been updated.

Case 3: If *T* is in *aborted* state, the database files are intact although some shadowed caches have been undone and others have not because abort command is not received.

Case 4: If *T* is in *committed* state, all HT-Log files have been written but the updates on the database file at *S* might not be finished.

From above analysis, it is known that only the committed transaction (Case 4) need be processed after site *S* is restarted. For the other cases, *T* is considered as being aborted.

To restart a crashed site *S*, first the most recent *checkpoint* record in the CT-Log file is located, and then the log file is scanned till end of file to find each committed transaction *T*. For each *T*, the corresponding HT-Log files are scanned to redo *T*. The recovery processing is much simple than that of the ordinary redo/undo method[11][16].

4.2 Communication Crash

To deal with communication crash, we adopted a simple approach. For an isolated site, it aborts all transactions from other sites, and aborts all transactions that have connections with other sites. However, for those transactions in the course of 2-phase-commit, a problem occurs when some sites have received the decision command, and some have not. To solve this problem, we extend 2PC protocol as follows. In *prepare phase*, each participant gets to know the other participants. In *decision phase*, if a participant can not get the decision from the coordinator, it then asks the other participants. If it can not communicate with any other participants, then the locks on these prepared pages must be held until the site can get the correct decision. Thus, the transaction is frozen at the site as shown in Fig. 3. The communication crash is detected by the timeout mechanism of WAKASHI[21].

5 Performance

This section shows a performance evaluation of the transaction management of WAKASHI. Our measurements are based on the OO7 benchmark[7], which presents a standard for benchmarking object-oriented database systems. The performance experiments were done by using the traversal operations of the OO7 benchmark. Three kinds of typical testing were conducted: read-only operation, commit operation, and abort operation. For last two experiments, the OO7 benchmark was extended to allow nested transactions.

5.1 Testbed

The Testbed consists of three SUN Sparc/20 workstations connected in an Ethernet network. The three machines have the same configuration with 4 Super-Spac+ processors(50MHz), 64 MB main memory, and a Segate ST31200N 1.05GB disk driver. The swapping space is 180 MB and the page size is 4KB. The operating system is SUN OS 5.3. The testing program was fetched from University of Wisconsin by ftp. We rewrote the programs by replacing definitions and dereferences of the persistent objects with INADA, which is an enhanced C++ language with the facilities for handling persistent objects[13]. Programs in INADA use WAKASHI to manage persistent objects.

The cold results were obtained by running the benchmark operations when the cache was empty at any testing machine. The hot results were obtained by repeatedly running the same benchmark operations. The time was computed by `gettimeofday()` function and the final results were the average of 4 runs of the benchmark operations.

Since the environment is better than those of other known testbeds, it is not fair to compare our result with others. We compare the results with those of the pure INADA programs, which used `mmap()` function instead of WAKASHI and ran at a local site, and check the overhead added by the transaction management system. We also compare the result of local executions with remote executions, and check the communication cost. All experiments were executed on three small OO7 databases that contained about 40,000 to 100,000 different objects. The database size corresponding to a fanout 3, 6, 9 was 12 MB, 14MB and 16MB, respectively. The experiments on medium OO7 databases (about 120MB) were also finished. Due to the limit of space, their results are omitted. We found that the results on small databases and medium databases were consistent.

5.2 Traversal Operation

Fig.4 shows the measurements of traversal operation T1 on a primary site and a mirror site. The former is a local operation and the latter is a remote operation. In case of the cold start, the response difference between the local operation and the remote operation is 300%. This is due to the cost on transferring and writing pages on the local caching disk. In case of the hot start, the result for the local operation is sped up 500% faster than that of the cold start, and the remote operation is sped up to the same response time as the local operation. This is due to the local disk caching function. In addition, the difference of the local operation with the pure INADA program is 1%. This shows the overhead on the locking is minimal because only at most 4000 page-locks are requested.

5.3 Transaction Commit Operation

Fig.5 shows the measurements of commit operation on the traversal T2C. These are hot results in order to make a clear comparison. A pure INADA result shows the time spent on updating the data and closing the file. A local operation is even faster than that of the pure INADA program because WAKASHI commits the transaction by generating the redo-only log

and doesn't need to flush the file. The remote operation is only 20% slower than the local operation

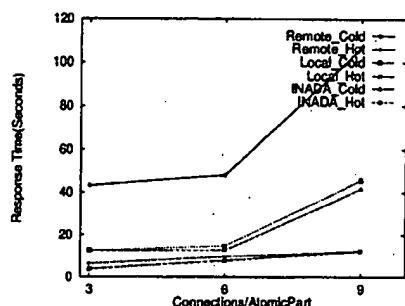


Figure 4: Result of Traversal Operation

because the result was committed with one communication operation by using the "compact commit" method. In addition, a nested T2C was implemented to test the parallel commit. Two T2C routines were nested in one transaction. They operated two remote OO7 databases respectively. The result is slower than that of the single remote operation, due to the serial execution of two T2C routines. The effect of parallel commit can be seen from the fact that the time is not doubled.

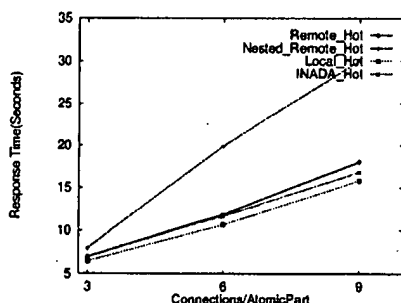


Figure 5: Result of Commit Operation

5.4 Transaction Abort Operation

Fig.6 reports the measurements of abort operation on the traversal T2C. For the same reason as the commit testing, only hot results are given. A pure INADA result shows the time spent on updating the data. A local operation is a little slower than the INADA operation. This is due to undoing the transaction by copying backup pages. The time of remote operation is similar to that of the local operation because no undo operation was performed in the primary site. Like the commit testing, a nested T2C was implemented to test the parallel abort. The result of the nested transaction is slower than that of the single remote operations, due to the serial execution of two T2C routines. The effect of parallel abort can be seen from the fact that the time is not doubled.

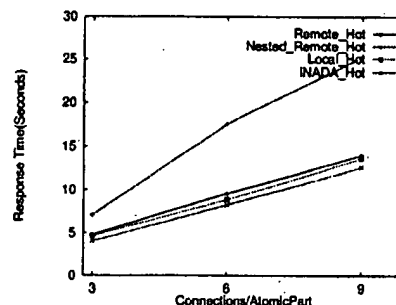


Figure 6: Result of Abort Operation

6 Related Work

In this section, we introduce comparisons with related work on the transaction management in the area of distributed object storage systems. Two known systems SHORE and Bess are chosen.

SHORE[6] is the latest version of object-oriented DBMS under development at the University of Wisconsin. The architecture is a symmetric peer-peer server like ours. We share the same approach on distributed commit protocol(2PC), concurrency control protocol(2PL) and coherency control protocol(Callback). However, because of the essential difference on the storage management between SHORE and WAKASHI (i.e., WAKASHI adopts memory-mapping approach while SHORE adopts ordinary buffer management approach), the locking mechanism is different in : (1) in WAKASHI, lock acquiring is transparent to the applications while SHORE requires applications to acquire locks by explicit function calls provided by the server; (2) in WAKASHI, the number of lock requests is equal to the number of pages accessed by a transaction while it is equal to the number of operations of the transactions in SHORE; (3) in WAKASHI, data and locks are cached in the server and can be shared by all clients that reside at this site while in SHORE each client cache allocates extra private space that is redundant if more than one client runs at a site. In addition, the commit and the recovery methods are different from us as stated below.

Bess[4] is a distributed server system developed by AT&T Bell Laboratories. It also adopted the same approach as SHORE but it improved the system efficiency by using the virtual memory techniques like ours. The differences between Bess and WAKASHI on transaction management are: (1) we adopt "multi-threaded-connection" approach not "single-threaded-connection" to execute transactions so that the parallelism of the transaction manager, transactions themselves, and multi-processors are exploited for high performance; (2) we employ a "compact commit" for the distributed commit while Bess commits page one by one that costs more communication; (3) we employ a "redo-only" method for crash recovery so that the logging cost is low and recovery is fast; by this approach only one pass is needed to restart from a crash un-